**What is Iterator Invalidation?**

An Iterator becomes invalidate when the container it points to changes its shape internally i.e. move elements from one location to another and the initial iterator still points to old invalid location.

Iterator invalidation in vector happens when,

* An element is inserted to vector at any location
* An element is deleted from vector.

**Iterator Invalidation Example on Element Deletion in vector:**

Suppose an iterator ‘**it’** points to a location x in the vector. Now suppose some deletion happens on that vector, due to which it move its elements from one location to another. Now if initial iterator ‘**it’** still points to old location then it becomes invalidated.

For example, in the below code we are deleting an element from vector using erase function. This erase function invalidates the current pointer. So if after calling the erase() function , if one uses the same invalidated iterator then it can result in undefined behavior.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | #include <iostream>  #include <vector>  #include <iterator>  #include <algorithm>    int main()  {      std::vector<int> vecArr;      for(int i = 1; i <= 10; i++)          vecArr.push\_back(i);        for(auto it = vecArr.begin(); it != vecArr.end(); it++)          std::cout<<(\*it)<<"  ";        std::cout<<std::endl;        // Erase and element with value 5.      auto it = std::find(vecArr.begin(), vecArr.end(), 5);      if(it != vecArr.end())          vecArr.erase(it);        // Now iterator 'it' is invalidated because it still points to      // old location, which has been deleted. So, if you will try to      // do the use the same iterator then it can show undefined      // behavior.        for(; it != vecArr.end(); it++)   // Unpredicted Behavior          std::cout<<(\*it)<<"  ";          // Unpredicted Behavior        return 0;  } |

Now, how to fix this ?

**Solution:**

After calling the erase function update the value of iterator **‘it’**i.e.

|  |  |
| --- | --- |
| 1  2  3  4 | // Erase and element with value 5.  auto it = std::find(vecArr.begin(), vecArr.end(), 5);  if(it != vecArr.end())     it = vecArr.erase(it); |

As, erase() function returns an iterator pointing to the new location of the element that followed the last element erased by the same function. Also, if the element deleted was the last element of the container then it returns the end of the container.

**Iterator Invalidation Example on Element Insertion in vector:**

When a new element is inserted in vector then it internally shifts its elements and hence the old iterators become invalidated.

Reasons for element shift are as follows,

* If element is inserted in between then it shift all the right elements by 1.
* If the new size of vector is more than its current capacity, then it relocates a bigger chunk of memory and copies all the elements there.

Therefore, when a new element is inserted in vector then its old iterator can become invalidated. Using this old invalidated iterators can result in undefined behavior i.e.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | #include <iostream>  #include <vector>  #include <iterator>  #include <algorithm>    int main()  {      std::vector<int> vecArr;      for(int i = 1; i <= 10; i++)          vecArr.push\_back(i);        auto it = vecArr.begin();      for(; it != vecArr.end(); it++)          std::cout<<(\*it)<<"  ";        std::cout<<std::endl;        it = vecArr.begin();        // Insert an element in position 2,      vecArr.insert ( it + 2 , 1 , 200 );        // Now old iterator it has become invalidated      // SO, using it as it is can result in undefined behavior        for(; it != vecArr.end(); it++)   // Undefined Behavior          std::cout<<(\*it)<<"  ";          // Undefined Behavior        return 0;  } |

Now, how to fix this ?

**Solution:**

After calling the insert function update the value of iterator **‘it’**i.e. by re-assigning it.

|  |  |
| --- | --- |
| 1  2  3  4  5 | // Insert an element in position 2,  vecArr.insert ( it + 2, 1 , 200 );    // Reinitialize the invalidated iterator to the begining.  it = vecArr.begin(); |